Efficient Distribution-Based Event Filtering

Annika Hinze, Sven Bittner
Freie Universität Berlin
{hinze,bittner@inf.fu-berlin.de}

Abstract

Event notification services are used in various applications, for example, stock tickers, environmental monitoring, and facility management. Several filtering algorithms for such services have been proposed. The best performance results are achieved by tree-based algorithms. However, to our knowledge existing algorithms do not consider the influence of event and profile distribution on the filter performance. In this paper we propose a distribution-dependent improvement of the tree-algorithm. We present the test results of our prototypical implementation that show the influence of various distribution-based measures on the performance.

1. Introduction

Event Notification Services (*ENS*) are used in various applications such as digital libraries, stock tickers, logistics, traffic control, collaborative work, facility management, remote monitoring, security, and project control. An ENS informs its users about new events that occurred on providers' sites. User interests are defined by means of profiles, which may consist of queries regarding primitive events, their time and order of occurrence, and of composite events, which are formed by temporal combinations of events. Various profile definition languages have been implemented, e.g., based on (attribute;value) pairs, SQL or XML-QL.

The observed events are filtered according to the user profiles. Several filter algorithms have been proposed, extensive evaluations of main-memory filtering algorithms are given in [6, 12]. The best performance is achieved by tree-based algorithms as introduced in [1, 8].

Run-time evaluations of ENS normally rely on test sets of equally distributed profiles and events. Our study of different scenarios shows that the assumption of equally distributed data does not hold for several applications. For example, in the case of stock tickers users are mainly interested in a small range of values for certain shares; the event data display high concentrations at selected values. Event data in environmental monitoring are recorded by sensors, equally distributed data are sensible. Nevertheless, users might be interested in catastrophe warnings, describing a small range of data of high importance.

In this paper, we introduce the theoretical foundation for a distribution-aware performance evaluation of tree-algorithms. The issues of event and profile distribution influencing the filter performance are examined. We propose an adaptive filter component that optimizes the profile tree for certain applications based on the data distributions. The optimization uses a restructuring of the profile tree. We introduce both an event-centric and a user-centric approach. Our argumentation is supported by selected results of the performance test of our prototype implementation.

The remainder of the paper is organized as follows: Section 2 describes related work. Section 3 reviews the concepts of event filtering. Section 4 introduces the distribution-based tree algorithm, briefly describes implementation details, and discusses selected results of our tests. Conclusions are drawn in Section 5.

2. Related Work

Event notification services support either subject- or content-based subscriptions (profiles). Subject-based services distribute events according to their topic or subject; content-based services filter events based on the content of event-messages. They support various kinds of profile queries, one of them are (attribute;value) pairs. These services are addressed in this paper.

Several filter algorithms have been developed for events and profiles defined in (attribute;value) pairs [3, 11, 13]. We distinguish three approaches: simple algorithms, clustering, and tree-based algorithms. Several hybrid algorithms have also been introduced; these algorithms support certain types of operators, such as equality- or inequality-operators, set-containment, or range-tests (see [6]). Other approaches support SQL-oriented queries [7, 10], XML-query languages [2, 4, 5], or IR-like keyword queries [15].

Our work is inspired by tree-based indexing strategies for keyword-based search, such as the ranked tree

Proceedings of the 22 nd International Conference on Distributed Computing Systems Workshops (ICDCSW'02)
0-7695-1588-6/02 $17.00 © 2002 IEEE

method [14]. Closely related to our work is the Elvin system [13] which includes a quenching mechanism that discards unneeded information without consuming resources. In Siena [3], the concept of early rejection on event-level is used for a distributed service. The service implements profile and event propagation within a network. Tree-based filtering approaches have been described in [1, 8, 9].

3. Concepts

This section is devoted to the presentation of the theoretical context of study. For illustration, we will use a toy example of an environmental monitoring system.

An *event* is the occurrence of a state transition at a certain point in time. In this paper, we only consider primitive events, each described as a collection of (attribute,value) pairs, such as the 3 pairs in the event

$$\begin{aligned} event(&temperature = 30\ ^\circ C,\\ &humidity = 90\ \%,\\ &radiation = 2\ mW/m^2) \end{aligned} \tag{1}$$

For a given application, we consider a firm set A of attributes a_j, with values belonging to given domains D_j. We denote the domain-size with $d_j \in \mathbb{N}$. For readability reasons, we use D instead of the correct D_j in general cases. A *profile* is a set of predicates also defined as (attribute,value) pairs. Profiles operate on the same attribute set as the events, not all attributes have to be specified.

$$\begin{aligned} profile(&temperature \geq 35\ ^\circ C,\\ &humidity = 90\ \%) \end{aligned}$$

The set of profiles defined in a particular ENS is denoted P, the number of profiles p. The number of attributes in events and profiles is n, $j \in [1, n]$. Considering profiles for value or range tests, each attribute's domain D is divided in, at the most, $(2p-1)$ subsets (referred to in the profiles) and an additional subset D_0 which is not referred to in any profile. Note that inequality tests can be translated to range tests. The $(2p-1)$ subsets are formed based on sets of non-overlapping subranges created from the, at the most, p different ranges defined in the p profiles [8]. In routing applications, each observed event is matched by at least one profile, and therefore $D_0 = \emptyset$. For filtering applications holds $D_0 \neq \emptyset$. We define $\overline{D} := D \setminus D_0$.

Our performance study is based on the fastest tree algorithm introduced in [8]: From a given set of profiles, a deterministic finite state automaton (DFSA) is created. The filtering is finally based on a profile tree of height n. The response time of a tree algorithm for range tests is bound by $O(nlog_2p)$ as shown in [6]. Each level $j \in [1, n]$ of the tree corresponds to the respective attribute a_j in the profiles. Edges starting at node a_j refer to attribute values defined in the profile set. For an observed event that matches one or more profiles, there is only a single path to follow in order to find the matched profiles. The following toy example serves as an illustrative reference throughout the paper.

Example 1 *Given is an environmental monitoring system: the service delivers sensor readings of temperature, humidity, and radiation. The attributes and their domains are:*

$$\begin{aligned} a_1:&\quad D_1 = [-30, 50]\ (temperature\ in\ ^\circ C)\\ a_2:&\quad D_2 = [0, 100]\quad (humidity\ in\ \%)\\ a_3:&\quad D_3 = [1, 100]\quad (UVA - radiation\ in\ mW/m^2) \end{aligned}$$

*In the following profiles $*$ denotes the fact that the user did not specify this attribute (*don't care *value).*

$$\begin{aligned} P1 &: profile(a1 \geq 35, a2 \geq 90, a3 = *)\\ P2 &: profile(a1 \geq 30, a2 \geq 90, a3 = *)\\ P3 &: profile(a1 \geq 30, a2 \geq 90, a3 \in [35, 50])\\ P4 &: profile(a1 \in [-30, -20], a2 \leq 5, a3 \in [40, 100])\\ P5 &: profile(a1 \geq 30, a2 \geq 80, a3 = *) \end{aligned}$$

The profile tree is depicted in Fig. 1. In the tree, $*$ refers to all *possible values, (*) denotes* all other *possible values of the given attribute. Note the subranges constructed from the overlapping profile ranges. Filtering the event defined in (1), the filtering path in the tree follows the edges $[30, 35) \rightarrow [90, 100] \rightarrow (*)$. The event is matched by the profiles $P2$ and $P5$.*

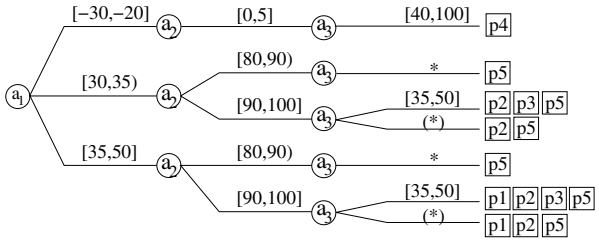


Figure 1. Profile tree for Example 1

The performance of the filter algorithm is measured in comparison steps (# operations), since the structure is stored in main memory. Both events and profiles have certain probability distributions for each attribute, given as continuous density functions (for continuous values) or discrete probability values (for discrete values). The response time of the filter is now described dependent on these distributions. We start by considering a sequential search in the profile values, then we evaluate the influence of other search strategies. We assume the profile values to be naturally ordered.

The distribution of the (continuous) event values of a certain attribute can be reformed as a distribution of, at the most, $(2p-1)$ discrete values, referring to the $(2p-1)$ subsets of $\overline{D}$. Thus, the probability of each subset is the sum

of the probability values for the values in the subset for the discrete case, and the integral of the density-function over the subset for the continuous case.

Each attribute value of an event is then modelled as the value of a discrete random variable X within an experiment. The domain W of X is described by the $(2p-1)$ subsets of $\overline{D}$. Additionally, x_0 refers to the subdomain D_0. The distribution of X is given as $(x_i, P_e(X = x_i)), x_i \in (W \cup \{x_0\})$, where x_i refers to the numbered subsets. The distribution of the values of a given attribute within the observed events is denoted by P_e, the similarly defined profile distribution as P_p. Formally, we define the zero-subdomain $D_0 \subseteq D$ as the set of values $v \in D$, which do not occur in the profile set. The probability of these attribute values is zero:

$$D_0 = \{v | v \in D, P_p(v) = 0\}$$

The domain-size of D_0 is denoted d_0. Since $W \cup x_0$ covers D, it follows $1 = \sum_i P_e(X = x_i) = \sum_i P_p(X = x_i)$.

If $D_0 = \emptyset$, then the number of filter operations for each attribute a can be described as the expectation for the random variable X. This correlation is essential for our studies described in this paper. The filter operations for $D_0 \neq \emptyset$ depend heavily on the filter algorithm. The response time $R(a, P_p, P_e)$ of a filter algorithm (measured in number of filter operations), where each attribute a is based on given distributions for profiles and events, can be expressed as

$$R(a, P_p, P_e) := E(X) + R_0(P_e, x_0) \qquad (2)$$

with

$$E(X) = \sum_{x_i \in W} x_i P_e(X = x_i) = \sum_{x_i \in W} x_i P_e(x_i)$$

The expectation $E(X)$ can vary according to the numbering order of the $x_i \in W$. If $D_0 \neq \emptyset$, the non-matching events have to be filtered, on average, in $R_0(P_e, x_0) = r_0 * P_e(X = x_o)$ operations, where r_0 refers to the number of steps needed to identify a non-matching event.

The distributions for the values of each of the n attributes of an event are not independent, the notion of conditional distributions is required in this context. The number of comparison steps for each of the n attributes in the profiles is then defined as the conditional expectations for $j \in [1, n]$

$$E(X_j | X_{j-1}, .., X_1) = \sum_{x_i^j \in W^j} x_i^j P_e(x_i^j | x^{j-1}, .., x^1)$$

The number of comparison steps for the profile matching of a certain event corresponds to an experiment with n random variables X_j, $j \in [1, n]$, which are not independent. If $D_0^j = \emptyset$ for all j, then the overall number of steps corresponds to the expectation

$$E(\sum_j (X^j | X^{j-1}, .., X^1)) = \sum_j E(X^j | X^{j-1}, .., X^1)$$

Thus, the response time R of a filter algorithm for all attributes can be expressed as

$$\begin{aligned} R &= \sum_{j=1}^{n} R(a_j, P_p^j, P_e^j) \\ &= \sum_{j=1}^{n} E(X^j | X^{j-1}, .., X^1) + \sum_{j=1}^{n} R_0(P_e^j, x_0^j) \end{aligned}$$

4. Distribution-dependent Tree Algorithm

This section introduces the distribution-based tree algorithm. We assume a history of profile and event distributions to be known to the system; the future properties of events and profiles are inferred from the history.

4.1. General Idea

The distribution-based algorithm evaluates first those event-values and attributes that have the highest selectivity. Selectivity can be defined for each attribute-value and for event-attributes, as we shall see. The tree is reordered such that attributes with high selectivity are at the top level of the tree, and for each attribute the values with highest selectivity are tested first.

Value selectivity
For each attribute we define a reordering of the values $x_i \in W$ as a function on the set of index-values $i \in [1, 2p-1]$: $o_v : [1, 2p-1] \rightarrow [1, 2p-1]$. The expectation of X is then

$$E(X) = \sum_{x_{o_v(i)} \in W} x_{o_v(i)} P_e(x_{o_v(i)})$$

The reordering is based on the value-selectivity, a function s_{val} defined on the attribute values $s_{val} : W \rightarrow \mathbb{R}$. The reordering follows the descending selectivity such that

$$\forall x_i, x_j \in W : s_{val}(x_i) \leq s_{val}(x_j) \Rightarrow o_v(i) > o_v(j)$$

The selectivity of values not contained in the profile tree is defined as zero. The order of values with equal selectivity is arbitrary (such as the natural order of the values). For the value-selectivity we can define various measures, for instance, the ranked tree method [14] uses IR-like ranking information for each keyword. The binary search defines another measure, as used in [1, 8]. It is not equally appropriate for all applications, as we will show in our tests. We propose three additional measures:

V1. Probability of the attribute values $P_e(x_i)$ according to event distributions: ordering of the profile values with descending $P_e(x_i)$.

V2. Probability of the attribute values $P_p(x_i)$ according to profile distributions: ordering of the profile values with descending $P_p(x_i)$.

V3. Probability of the attribute values according to event and profile distributions: ordering of the profile values with descending $P_e(x_i) * P_p(x_i)$.

The influence of the reordering on the expected value is demonstrated in the next example:

Example 2 *Let us assume the following probabilities for the values of attribute a_1(temperature) and the resulting reordering according to Measure V1.*

$$\begin{array}{rlll} x_1 = [-30,-20] & P_e(x_1) = 2\%, & o_v(1) = 2 \\ x_2 = [30,35] & P_e(x_2) = 1\%, & o_v(2) = 3 \\ x_3 = (35,50] & P_e(x_3) = 80\%, & o_v(3) = 1 \\ x_0 = [-20,30] & P_e(x_0) = 17\% \end{array}$$

The expected value is

$$\begin{aligned} E(X^1) &= \sum_{x^1_{o(i)} \in W} x^1_{o(i)} P_e(x^1_{o(i)}) \\ &= x^1_{o(1)} P_e(x^1_{o(1)}) + x^1_{o(2)} P_e(x^1_{o(2)}) + x^1_{o(3)} P_e(x^1_{o(3)}) \\ &= x^1_2 P_e(x^1_2) + x^1_3 P_e(x^1_3) + x^1_1 P_e(x^1_1) \\ &= 0.02 * 2 + 0.01 * 3 + 0.8 * 1 = 0.87 \end{aligned}$$

The number of filtering operations to identify non-matching events depends on the implementation. In our prototype, a non-match is discovered after the number of steps that would have been needed to identify the requested value in the tree. It then follows that $R = 0.87 + 2 * 0.17 = 1.21$. Reordering according to binary search leads to the expectation $E(X^1) = 0.01 * 1 + 0.02 * 2 + 0.8 * 2 = 1.65$. The identification of non-matches takes $r_0 = log_2(2p-1)$ steps, this leads to $R_0 = 2 * 0.17 = 0.34$ and therefore $R = 1.99$ in our example. In the following examples we will only discuss the expected values for successful matches.

Attribute selectivity

We define a reordering of the attributes $a_j, j \in [1,n]$ as function on the attribute indexes: $o_a : [1,n] \rightarrow [1,n]$. The expectation of the n random variables $X_j, j \in [1,n]$ is then

$$E(\sum_{o(j)}(X^{o(j)}|X^{o(k)}, k \leq o(j))) = \sum_{o(j)} E(X^{o(j)}|X^{o(k)}, k \leq o(j))$$

The reordering o_a is based on the attribute-selectivity $s_{att} : A \rightarrow \mathbb{R}$, it follows the descending selectivity such that

$$\forall a_i, a_j \in A : s_{att}(a_i) \leq s_{att}(a_j) \Rightarrow o_a(i) > o_a(j)$$

We propose three measures for the attribute selectivity: the first considers only the attribute domains and profile distribution, the second also takes the event distribution into account, the third measure depends on the conditional distributions of the profiles - the shape of the tree.

A1. For each attribute the ratio of the size of the zero-subdomain regarding the profile distribution and the domain-size: $s_{att}(a_j) = \frac{d_0(a_j)}{d_j}$.

A2. The ratio of the probability of the zero-subdomain and the probability of the domain-size under consideration of the profile distribution: $s_{att}(a_j) = \frac{d_0(a_j) * P_e(D_0(a_j))}{d_j}$. $P_e(D_0)$ denotes the probability that the data of an event take attribute values of D_0: $P_e(D_0) = P_e(X = x_0)$.

A3. The realtive size of the zero-subdomains D_0 depending on the conditional probabilities. These probabilities influence the shape of the tree. The attributes have to be ordered in the tree such that the sum of the zero-subdomains is maximal. The determination of this function is expensive with $O(n! * (2p-1))$.

The examples show the influence of the first two measures:

Example 3 *Derived from the sizes of the attribute domains defined in Example 1 the selectivities of the attributes based on Measure A1 are:*

$$\begin{array}{lllll} a_1: & d_1 = 80 & d_0 = 50 & s_{att}(a_1) = 0.625 \\ a_2: & d_2 = 100 & d_0 = 75 & s_{att}(a_2) = 0.75 \\ a_3: & d_3 = 100 & d_0 = 0 & s_{att}(a_3) = 0 \end{array}$$

We assume the distributions $P_e(X^1)$ as in Example 2, and

$$\begin{aligned} P_e(X^2) = ([0-30]:5\%, [30-80]:60\%, \\ [80-90]:25\%, [90-100]:10\%), \text{ and} \end{aligned}$$

$$\begin{aligned} P_e(X^3) = ([0-35]:90\%, [35-40]:5\%, \\ [40-50]:2\%, [50-100]:3\%). \end{aligned}$$

For ease of computation we assume independent attributes. The expected value for the original tree (Fig. 1) is

$$\begin{aligned} E(\sum_{j=1}^{3}(X^j|X^{j-1},..,X^1)) &= \\ E(X^1) + E(X^2|X^1) + E(X^3|X^2,X^1) &= \\ 2.44 + 0.568 + 0.363 &= 3.371 \end{aligned}$$

Reordering according to Measure A1 leads to

$$\begin{aligned} E(\sum_{o_a(j)=1}^{3}(X^o_a(j)|X^{o_a(j)-1},..)) &= \\ E(X^2) + E(X^1|X^2) + E(X^3|X^1,X^2) &= \\ 0.85 + 0.364 + 0.702 &= 1.91 \end{aligned}$$

Reordering based on Measure A2 for the selectivities of the attributes leads to the same result: $s_{att}(a_1) = 0.05$, $s_{att}(a_2) = 0.6$, $s_{att}(a_3) = 0$.

Applying both the value-based and the attribute-based reordering is briefly shown in the next example:

Example 4 *The expected value for a tree using the Measures V1 and A2 is* $E(\sum_{o_a(j)=1}^{3}(X^{o_a(j)}|X^{o_a(j)-1},..)) = 0.50 + 0.285 + 0.300 = 1.08$ *The resulting tree is depicted in Fig. 2. Binary search in the attribute-reordered tree leads to* $E(\sum ..) = 1.616$.

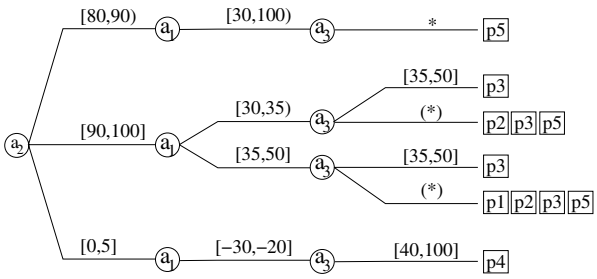


Figure 2. Reordered profile tree (Example 4)

4.2. Implementation

We implemented a prototypical system for the evaluation of algorithm variations. The system is implemented as a generic service: all events, attributes, domains, and compare operators can be created and specified at runtime. Similarly to [8] we use a tree structure for the matching. Currently, our prototype supports only equality tests and *don't* care cases. Range tests are not necessary for our estimation of comparison steps.

Creating the tree
The prototype supports the following value orders (either descending or ascending): natural order as implied by domain, profile-dependent probability, event-dependent probability and combined probability in profiles and events. If the attribute values in the tree are not stored in the predefined order, we use a lookup table to keep the order of the elements in the tree. The table contains a position for each element, where position relates to the reference of the value in the defined order. Instead of using such tables, the elements themselves could represent their position, but our implementation focuses on more general classes for elements.

Searching the tree
We implemented two search strategies for the tree: (1) following the edges in the defined order, or (2) using binary search on the given order. Linear search based on values in ascending order follows the rule: A tree node does not contain the searched value if a greater one is found. This can also be applied for probability-based order since the lookup table with the position information is also stored in the tree. After finding a value with higher probability than the one of the requested value, the search is stopped. This rule prevents the search through the complete node. Descending order works analogously.

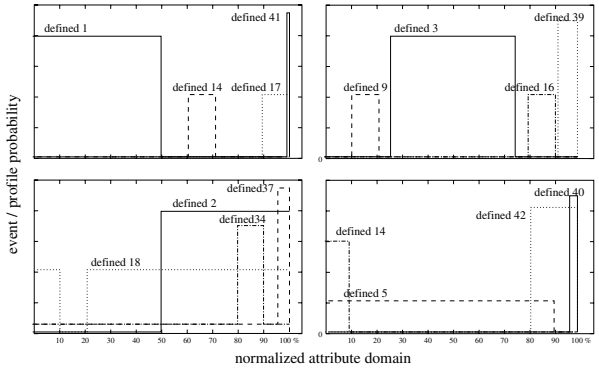


Figure 3. Exemplary distributions

Example 5 *This example demonstrates the linear table-based search:*
Domain: $\{a, b, c, d, e, f\}$*, Order of elements:* f, c, a, b, e, d
Table: $\{('f', 1), ('c', 2), ('a', 3), ('b', 4), ('e', 5), ('d', 6)\}$
Tree-order: f c b e d
When searching for value $'a'$ *we can stop at* $'b'$ *since* $4 > 3$ *indicates that the tree does not contain value* $'a'$ *for that particular node.*

Statistics
We implemented statistic objects with counters for events, attributes, operators, and values If a profile specifies a certain value that element-counter is incremented. For the tests, we manipulate the counters in order to simulate a distribution. In that way for a test all events and profiles are created only once, the statistic objects are initialized for chosen distributions, and the tree is built with defined order of the edges. Then, all events are posted and the average number of visited edges is determined based on the distribution defined in the statistic objects. The result is similar to posting the events with the given distribution (which requires a multiple number of events). The prototype also supports the direct event matching without simulating.

4.3. Test Results and Discussion

Within our performance tests we defined 60 distributions for events and profiles. Within this paper we can only show a selection of our results. Fig. 3 displays the distributions we refer to within the discussion of the results. The graphs do not precisely describe each function, but give an impression of the distribution. Additionally, we use equally distributed data and the Gauss-distribution.

Value-reordering
We tested all permutations of the 60 distributions with 8 different orderings plus binary search in 4 test scenarios:

TV1. Creation of profile tree (n attributes), 10,000 profiles according given distribution, event tests until 95% precision for average #operations is reached.

TV2. Full profile tree (n attributes), event tests until 95% precision for average #operations is reached.

TV3. Full profile tree with one attribute only, 4000 events according given distribution.

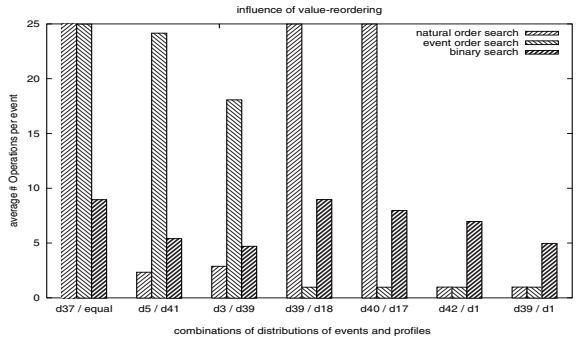
TV4. Full profile tree with one attribute only, all possible events, average #operations computed based on #operations and event distribution (according to Eq. 2).

Fig. 4(a) displays selected results for different P_e/P_p combinations. The data for three value orderings are presented: natural order, descending event distribution (Measure V1), and binary search. Natural and event-based ordering have oscillating response time, where binary search provides balanced results. The figure demonstrates that there is no "perfect" approach; depending on the distributions, different ordering strategies provide best performance. There are typical distributions of events and profiles for certain applications and for these applications special filter approaches are appropriate. The ordering according to event distribution shows best performance for distributions with *peaks*: a small range of events is requested by many users, while a wide range of events is not matched by the profiles (a scenario for catastrophe warning systems).

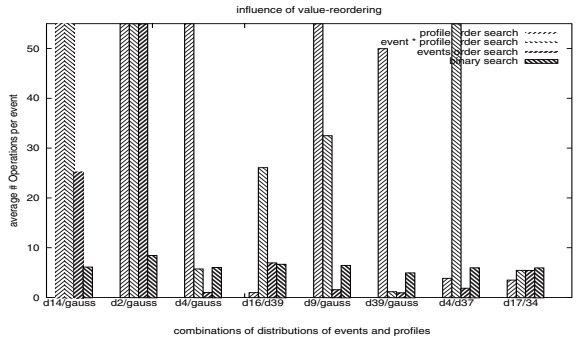
Formally, event-based order is faster than binary search if $E(X) < log_2(2p-1)$, assuming similar strategies for non-matching events. The selectivity based on the event order is a fragile measure, not robust to changes in the distributions. Reordering based on this measure is therefore recommended for systems with stable distributions.

Fig. 4(b) displays the results of reordering based on Measures V1-V3. Again, for selected cases the reordering according to event distribution provides best performance. The profile-based reordering (V2) supports certain profiles which leads to a decreasing average performance with respect to the events. The reordering based on Measure V3 follows a middle course: frequent events that are of high interest for users are supported, the average performance is lower than in the event-based order but events that have no subscribers are postponed. Both the profile dependent measures support user groups with similar interest.

Fig. 5 shows the tests results where the performance is measured as average comparison steps per event(Fig. 5(a)), per profile (Fig. 5(b)), and with respect to events and users (Fig. 5(c)). The tests have been performed using equally distributed events, events with descending probability and



(a) Based on Measure V1, (TV4)



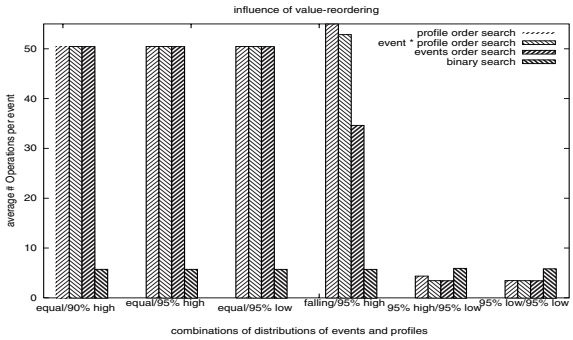
(b) Measures V1-V3 vs binary search, (TV4)

Figure 4. Value reordering, Measures V1-V3

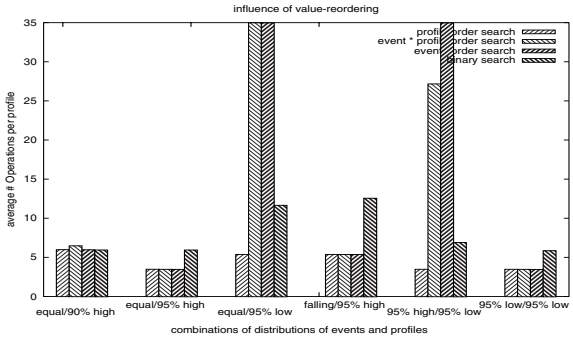
equally distributed events with a small peak. The profiles are equally distributed with a small peak, the number refers to the probability of the peak-values. *High* and *low* refers to the location of the peak at the lower or higher values.

We selected these distributions to discuss the influence of reordering according to the profile-dependent Measures V2 and V3. The results of the first two tests (blocks 1 and 2) in Fig. 5 indicate that the value selectivity is a fragile measure that is influenced by light changes in the event/profile distribution. For equally distributed events, different profile distributions did not influence the event response time, but the profile response time (see data block 2 and 3). Here, the profile-dependent reorderings (V2 and V3) significantly improve the performance per profile (block 3). The blocks 5 and 6 show that the profile response time differs also for distributions with an overall minimal event response time.

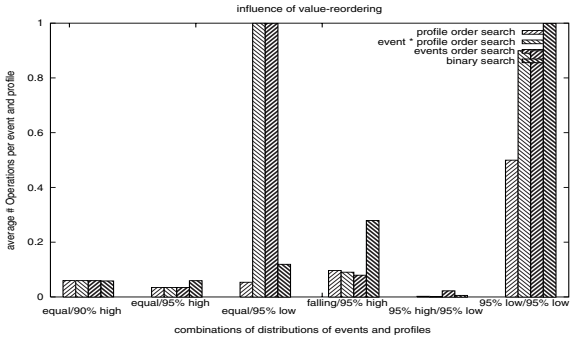
In summary, algorithms based on V2 and V3 lead to inferior average response time according to the events, but to faster notifications for profiles with high priority. For filter components operating in their optimal working point ($freq_{events} \ll freq_{filter}$) events do not queue. Thus, our algorithm improves performance for selected profiles since fast filtered events are not slowed down by other events.



(a) Average filter operations per event



(b) Average filter operations per profile



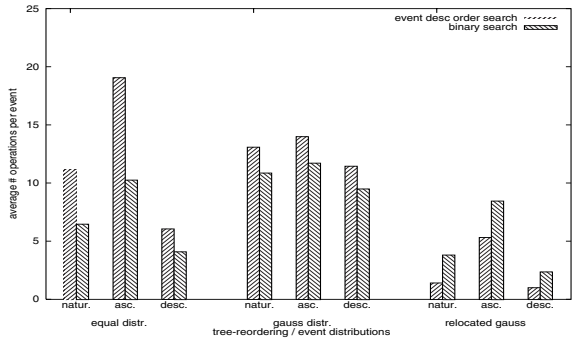
(c) Average filter operations per event and profile

Figure 5. Value reordering (TV4)

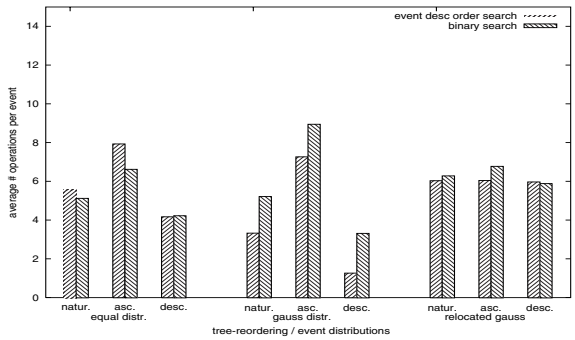
Attribute-reordering

For selected profile distributions, different attribute orderings have been tested. The influence of the Measures A1 and A2 has been tested by applying three different event distributions: equal, Gauss, and relocated Gauss (center of the distribution shifted to the low and high values).

Here, we discuss the results of two experiments. For each experiment, the profile tree contains 5 attributes with different selectivities according to Measure A1 and A2.



(a) Experiment TA1: wide differences in attribute distributions



(b) Experiment TA2: small differences in attribute distributions

Figure 6. Attribute reordering (TA1 and TA2)

TA1. The first uses distributions with peaks of width from 10%-80% (Fig. 6(a)).

TA2. The second experiment uses distributions that only lightly vary (Fig. 6(b)).

The three tree-orderings refer to the natural order of the attributes according to their index-number (natur.), ascending order (asc.) and descending order (desc.) according to the attribute selectivity. We show the results of both selectivity-based and binary search. The data for binary search vary for the different orderings, since non-matching events are detected differently. Note that the ascending order describes the worst-case scenario. Assuming independence of the attributes, the tests have been performed using the overall distribution of events for each attribute, not conditional distributions. Fig. 6(a) (left) displays the data for equally distributed events. Here, the ascending event-based order has better results than the descending order owing to the predefined internal order of the events values in case of equal event distributions. Fig. 6(a) (center) displays the results of a similar test with Gauss-distributed events, the reordering according to Measure A2 has a stronger influence on the performance. The events relating to the zero-

subdomain have to be dismissed as early as possible. Therefore, the corresponding attributes reside at the upper levels of the tree. The reordering is not faster than the binary search since $E(X) > log_2(2p-1)$. Fig. 6(a) (right) refers to a similar profile tree. We now consider a relocated Gauss-distribution that features especially high probability for events referring to zero-subdomains. Reordering of the tree shows best performance if ordered according to Measure A2. The reordering is faster than binary search since a significant part of the events map onto the zero-subdomain.

If we used Measure A1, the three tests would lead to similar results for all event-distributions. In all three test-series, the reordering provides better performance; the influence is most significant in the third test-case.

Fig. 6(b) displays the data for equally distributed events (left), for events distributed according to Gauss (center) and to a relocated Gauss (right). Similar patterns can be found, the ascending event based reordering improves the filter performance. In this experiment, the relocated Gauss-distribution leads to an overall lower performance.

In summary, attribute reordering according to Measures A1-A3 can be used to reject unmatched events early: Reordering with Measure A1 is useful for equally distributed data, Measures A2 and A3 are appropriate for all event distributions. Measure A3 is costly to obtain and therefore only sensible for applications with stable distributions. Our theoretical model is supported by exemplary tests.

5. Conclusion & Outlook

In this paper, we proposed a distribution-dependent improvement of the fastest filtering algorithm [8] for Event Notification Services. Both value-dependent and attribute-dependent selectivity measures are introduced as basis for the improved algorithm. We have shown analytically that our distribution-based approach improves the average case performance of tree-based algorithms. The algorithm has also been implemented and extensively tested with simulated event and profile distributions. A selection of the test results is discussed in the paper. The algorithm can either work based on predefined distributions for the observed events, or it has to maintain a history of events in order to determine the event distribution.

Event filtering algorithms should be adaptive in order to apply the optimal filtering strategy for each attribute. Sensible strategies are selectivity-based reorderings of attributes and values, binary-, interpolation-, or hash-based search within attribute-values.

We are currently implementing the prototype of a generic parameterized Event Notification System (*GENAS*) that is based on the filter algorithm introduced here. We will extend the filter to handle composite events. We also investigate the influence of *don't care*-edges and different operators on the performance.

Our approach can be used to reduce workload in resource critical environments, for example in mobile computing. Unnecessary event information is rejected as early as possible and the filtering can be optimized according to the application and user needs.

Acknowledgements: We wish to thank the database group at the FU Berlin and the reviewers for their valuable comments on our approach.

References

[1] M. Aguilera, R. Strom, D. Sturman, M. Astley, and T. Chandra. Matching events in a content-based subscription system. In *SIGMOD Distributed Computing*, 1999.

[2] M. Altinel and M. J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *VLDB*, 2000.

[3] A. Carzaniga, D. Rosenblum, and A. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001.

[4] J. Chen, D. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A scalable continuous query system for internet databases. In *SIGMOD Management of Data*, 2000.

[5] F. Fabret, H. Jacobsen, F. Llirbat, J. Pereira, and D. Shasha. Webfilter: A high-throughput xml-based publish and subscribe system. VLDB, Rome, Italy, 2001.

[6] F. Fabret, F. Llirbat, J. Pereira, and D. Shasha. Efficient matching for content-based publish/subscribe systems. Technical report, INRIA, 2000.

[7] S. Gatziu and K. Dittrich. SAMOS: An Active Object-Oriented Database System. *IEEE Quarterly Bulletin on Data Engineering*, 15(1-4):23–26, 1992.

[8] K. J. Gough and G. Smith. Effizient Recognition of Events in a Distributed System. In *Proceedings of the ACSC-18*, 1995.

[9] E. N. Hanson, M. Chaabouni, C. Kim, and Y. Wang. A predicate matching algorithm for database rule systems. In *Proc. of SIGMOD'90*, pages 271–280, 1990.

[10] L. Liu, C. Pu, W. Tang, and W. Han. Conquer: A Continual Query System for Update Monitoring in the WWW. *International Journal of Computer Systems, Science and Engineering*, Special issue on Web semantics, 1999.

[11] J. Pereira, F. Fabret, H. Jacobesen, F. Llirbat, R. Preotiuc-Prieto, K. Ross, and D. Shasha. Le subscribe: Publish and subscribe on the web at extreme speed. In *SIGMOD*, 2001.

[12] J. Pereira, F. Fabret, F. Llirbat, and D. Shasha. Efficient matching for web-based publish/subscribe systems. In *CooPIS*, 2001.

[13] B. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *AUUG97*, 1997.

[14] T. W. Yan and H. García-Molina. Index structures for selective dissemination of information under the Boolean model. *ACM Trans. Database Syst.*, 19(2):332–334, 1994.

[15] T. W. Yan and H. García-Molina. SIFT - a tool for wide-area information dissemination. In *USENIX*, 1995.